**NTUST OOP Midterm Problem Design**

|  |
| --- |
| **Subject:** **Square and Multiply** |
| **Author: 廖聖郝 (SHENG-HAO LIAO)** |
| **Main testing concept:** 大數運算、二進制轉換、演算法   |  |  | | --- | --- | | **Basics** | **Functions** | | ■ C++ BASICS  ■ FLOW OF CONTROL  □ FUNCTION BASICS  □ PARAMETERS AND OVERLOADING  □ ARRAYS  □ STRUCTURES AND CLASSES  □ CONSTRUCTORS AND OTHER TOOLS  □ OPERATOR OVERLOADING, FRIENDS,AND REFERENCES  □ STRINGS  □ POINTERS AND DYNAMIC ARRAYS | □ SEPARATE COMPILATION AND NAMESPACES  □ STREAMS AND FILE I/O  □ RECURSION  □ INHERITANCE  □ POLYMORPHISM AND VIRTUAL FUNCTIONS  □ TEMPLATES  □ LINKED DATA STRUCTURES  □ EXCEPTION HANDLING  □ STANDARD TEMPLATE LIBRARY  □ PATTERNS AND UML | |
| **Description:**  In computers, due to the issue of large numbers and performance, we cannot use regular calculation methods to handle the above equation. However, there is a common algorithm called "Square and Multiply" that can quickly calculate (b is large number).  Square and Multiply Algorithm:   1. Transfer large number b into binary format.   e.g.: 20(10) → 10100(2),  8943793798137911527249106497563(10) → 1110000111000101111010011010101000111100010110011011100011101011110100011111010010110001111000000011011(2)   1. You have a variable called *Result*, and initial with value 1. 2. Scan the binary from left to right, and do: 3. If digit is 1 execute <square>, then execute <multiply> 4. If digit is 0 execute <square>   e.g.: 10100(2) → <square><multiply>, <square>, <square><multiply>, <square>, <square>  1    0  0  1  0  1   1. After all iteration *Result* variable is the value of   Operations:  <square>: square *Result* and mod p, then store back into *Result*.  <multiply>: multiply *Result* by a and mod p, then store back into *Result*.  **Input:**  Each test cases may contain multiple inputs and outputs. Each line contains three positive integers a, b and p (a < 100000, b is a very large number that cannot be represented by a built-in data type, p < 10000).  a b p  **Output:**  Value of for every inputs.  **Sample Input / Output：**   |  |  | | --- | --- | | Sample Input | Sample Output | | 2 5 7  3 45 7  2447 5992 873  7414 8989898989898989898989898989 9453  47 8943793798137911527249106497563 159 | 4  6  1  4537  119 | |
| **■ Eazy,Only basic programming syntax and structure are required.**  **□ Medium,Multiple programming grammars and structures are required.**  **□ Hard,Need to use multiple program structures or more complex data types.** |
| **Expected solving time:**  20 minutes |
| **Other notes:**  This is an algorithm commonly used in asymmetric encryption/decryption. |